«Основные алгоритмы»

ЗАДАЧИ, УКАЗАНИЯ, РЕШЕНИЯ И КРИТЕРИИ ПРОВЕРКИ

итоговой контрольной 2018

В этом файле приведены условия задач итоговой контрольной, критерии их проверки, а также указания и решения некоторых задач. Приведённые критерии описывают общие случаи и не являются исчерпывающими. Если ваш случай не попадает под критерии, значит задача оценена исходя из общих соображений, согласованных с критериями.

Указания к задаче содержат основные шаги её решения и не являются полным решением. В частности, в них часто опущены обоснования приведённых фактов или доказательство оглашённых утверждений. Подобный подход к решению задачи студентом в большинстве задач приведёт к неполному баллу за её решение.

В случае, если для задачи был предложен правильный алгоритм, то отсутствие доказательства корректности и оценки его сложности существенно снижает оценку за решение задачи как и неоптимальность алгоритма, о чём было сообщено в преамбуле к контрольной.

Преамбула к контрольной

Необоснованные ответы не оцениваются! Если в задаче требуется построение алгоритма, то нужно построить оптимальный алгоритм (за неэффективность снижается оценка), доказать его корректность и оценить время работы. Задачи расположены в порядке тем курса.

Введение. Жадные алгоритмы

Задача 1 (3). На вход задачи подаётся последовательность целых чисел (диапазона integer) $x_1, \ldots, x_n$, которая оканчивается нулём (0 встречается ровно один раз). Постройте онлайн алгоритм, который находит максимальное произведение $x_i \times x_j$, такое, что $|i - j| < 8$, и использует для этого $O(1)-$памяти.

Критерии.

-3 алгоритм или был не онлайн, или требовал $O(n)$ памяти, или не учитывал условие $|i - j| < 8$;

-1 алгоритм не учитывал минимальные элементы, а использовал только 2 максимальных элемента для произведения;

-1 не оптимальный алгоритм - работает медленнее, чем $O(n)$.

Указания. Алгоритм хранит в памяти последние 7 чисел и максимальное произведение на текущем шаге. При получении нового числа последовательности, алгоритм обновляет максимальное произведение, если x_i в паре с каким-то числом в памяти образует произведение, большее, чем лежит в памяти. □

Оценка сумм. Рекурсия и итерация

Задача 2 (3). Определим $f(n)$ как количество выводов «Hello, World!» следующей функцией (на входе n).

```
1  Function HelloWorld(n) :
2      if n < 2018 then
3          for i = 1 to n do
4              print("Hello, World!");
5          end
6      else
7          HelloWorld(⌊n/3⌋);
8          print("Hello, World!");
9          HelloWorld(⌊n/3⌋);
10         for i = 1 to 2018 do
11             print("Hello, World!");
12         end
13     end
14 end
```

Оцените асимптотику роста $f(n)$.

Критерии.

0 выписана неверная рекуррента;

$+0,5$ выписана верная рекуррента;

$-1,5$ считалось, что в строках 3-5 `print("Hello, World!")` вызывается $\Theta(n)$ раз и это переходило и в общую формулу;

$-0,5$ O-оценка вместо Θ-оценки, а также другие не слишком значительные ошибки.

Указания. Для f справедлива рекуррентная формула $f(n) = 2f(\lfloor \frac{n}{3} \rfloor) + \Theta(1)$, что позволяет оценить f по основной теореме о рекурсии. □

Замечание. При решении задачи студенты часто писали, что $f(n) = \Theta(n)$ при $n < 2018$. Это высказывание абсурдно, потому что с тем же успехом можно было бы написать, что $f(n) = \Theta(2^n)$, поскольку число n ограничено константой. При малых n важно, что $f(n) = \Theta(1)$ (а не $o(1)$, например). Часто это приводило к ошибке из критериального случая на $-1,5$.

Алгоритмы «разделяй и властвуй»

Задача 3 (2+3). Найдите лучшие верхние и нижние оценки на функции, считая, что при малых n, $T(n) = \Theta(1)$:

а) $T(n) = 27T(\lfloor n/3 \rfloor) + \frac{n^5}{n^2+\log n}$;

б) $T(n) = T(\frac{n}{6}) + T(\frac{5n}{9}) + 2n$.

Критерии.

Пункт а)

+1 Показано, что выполняются условия II пункта основной теоремы для рекуррентных соотношений, в частности пояснено, почему $f(n) = \Theta(n^l)$, где l зависит от номера варианта.

+1 Получен правильный ответ при условии хотя бы частичного выполнения первого пункта.

Пункт б)

Правильный ответ - $T(n) = \Theta(n)$.

Есть два альтернативных доказательства, у которых в совокупности практически монополия. Имеет смысл оценивать решения согласно тому, к какому типу они принадлежат.

1. **Обычное (счетное) решение**

   +1 Найдена или оценена минимальная или максимальная глубина дерева рекурсии или приведен рисунок дерева с комментарием о глубине.

   +1 $T(n)$ представлена в виде суммы по дереву до некоторого уровня.

   +1 Установлено, что эта сумма сворачивается во что-то в духе Cn.

   −1 Не показано, что $T(n) = \Omega(n)$.

   −1 Не показано, что $T(n) = O(n)$.

2. **Индуктивно-подстановочное решение**

   +1 Сделан корректный, обоснованный индуктивный переход.

   +1 Показано, что $T(n) = \Omega(n)$.

   +1 Показано, что $T(n) = O(n)$.

Сортировки и нижние оценки

Задача 4 (4). A — строго возрастающий массив длины n. Функция $f(x)$ возвращает $A[x] < \infty$ при $1 \leqslant x \leqslant n$ и ∞ при $x > n$. Постройте алгоритм, который проверяет, встречается ли y среди значений f за $O(\log n)$, вызывая функцию f как чёрный ящик.

Критерии.

+1 Упомянут бинпоиск или записано его верное описание в верном контексте и без неверных уточнений

−1 Асимптотика алгоритма в решение $O(logn + logx)$, где x – взятое неудачным образом первое приближение n

−1 Нет доказательсва асимптотики

Указания. Будем вычислять $f(2^k)$ пока не получим, что $f(2^k) = \infty$. Это значит, что $2^{k-1} \leqslant n < 2^k \leqslant 2n$, отсюда $k = O(\log n)$. Осталось проверить, встречается ли y среди значений f при $1 \leqslant x \leqslant 2^k$ бинарным поиском, который также стоит $O(\log n) = O(k)$. □

Задача 5 (2+5). На вход подается число n и n пар целых чисел

$$\{(x_1, y_1), \ldots, (x_n, y_n)\}.$$

Эти точки являются парой "аргумент-значение" для некоторой функции $y = f(x)$, $f : \mathbb{Z} \to \mathbb{Z}$. Точки подаются на вход в произвольном порядке. Постройте алгоритм, определяющий может ли f быть

а) строго монотонной; **б)** выпуклой?

Критерии. [Пункт а]

−1 нет доказательства корректности;

−0,5 получено только необходимое условие существования функции, но не доказано существование при положительном ответе;

−0,5 нет оценки на время работы;

-2 нет обоснования алгоритму или алгоритм отсутствует.

Критерии. [Пункт б]

$-4,5$ нет доказательства корректности;

-4 доказано почему последовательность монотонна, но нет обоснования (не)существования искомой функции;

$-3-1$ приведены идеи, связывающие выпуклость последовательности с выпуклостью функции (в зависимости от грамотности и полноты изложения);

-2 написано слово «производная»;

$-0,5$ получено только необходимое условие существования функции, но не доказано существование при положительном ответе;

$-0,5$ нет оценки на время работы;

-5 нет обоснования алгоритму или алгоритм отсутствует.

Решение. Будем считать, при доказательстве корректности, что имеем дело с функцией $f : \mathbb{R} \to \mathbb{R}$, определённой в целых точках, в которых она принимает целые значения. На монотонность и выпуклость это допущение не влияет.

Проверка строгой монотонности.

Отсортируем массив точек $Dots$ по y по возрастанию. Проверяем онлайн является ли массив $Dots$ отсортированным по x и имеются ли в нем одинаковые значения по y. Если массив оказался отсортирован по x по возрастанию и в нем нет точек с одинаковыми значениями y, выводим "функция может быть строго возрастающей". Если массив оказался отсортирован по x по убыванию и в нем нет точек с одинаковыми значениями y, выводим "функция может быть строго убывающей". В остальных случаях выводим "функция не может быть строго монотонной".

Корректность. В случае когда массив $Dots$ отсортирован достаточно рассмотреть кусочно-линейную функцию, последовательно соединив соседние точки массива $Dots$, которая является строго монотонной при отсутствии последовательных точек с одинаковыми ординатами. Если

массив не отсортирован, то существует такая тройка точек d_1, d_2, d_3 что выполняется

$$\begin{cases} d_1.x < d_2.x < d_3.x \\ d_1.y < d_2.y > d_3.y \textbf{ или } d_1.y > d_2.y < d_3.y \end{cases}$$

Что противоречит свойству монотонной функции: $\forall x_1 > x_2 \ f(x_1) > (<) f(x_2)$.

Время работы. Вся работа алгоритма - сортировка по y и затем проход по массиву в ходе которого делается константное число сравнений. Суммарно - $O(n \log n) + O(n) = O(n \log n)$.

Проверка выпуклости.

Отсортируем массив точек $Dots$ по x. Построим по нему массив $K[1..n-1]$ следующим образом:

$$K[i] = \frac{Dots[i+1].y - Dots[i].y}{Dots[i+1].x - Dots[i].x}.$$

Заметим, что знаменатель в формуле не может быть равен нулю т.к. точки $Dots$ - точки непрерывной функции. После чего пробегаем по массиву K считая сколько раз рост значений сменялся на убывание. Для этого нужен один счетчик $ChangesCount$, одна индикаторная переменная $ChangesSign \in \{\pm 1, 0\}$ и сравнение соседних элементов:

$ChangesCount := 0;$
$ChangesSign := 0;$
for $i = 1$ **upto** $n - 2 : \{$
  if $Sign(K[i+1] - K[i]) + ChangeSign == 0 \textbf{and} ChangeSign \neq 0$ **then** $\{$
    $ChangesCount++; ChangeSign := Sign(K[i+1] - K[i]); \}$
  elseif $Sign(K[i+1] - K[i]) \neq 0$ **then** $ChangeSign := Sign(K[i+1] - K[i]);$
  $i++;$
$\}$

Если $ChangesCount = 0$ то функция может быть выпуклой, в противном случае нет. Если помимо $ChangesCount = 0$ $ChangeSign = 0$ выдаем, что функция может быть выпуклой как вверх так и вниз, $ChangeSign = 1$ - выпукла вниз, $ChangeSign = -1$ - выпукла вверх.

Корректность. Алгоритм проверяет на выпуклость кусочно-линейную функцию $f(x)$, полученную последовательным соединением соседних точек отрезками прямой. Массив K - это массив углов наклона линейных компонент $f(x)$. Если K - состоит из одного элемента, то в данном случае $f(x)$ - прямая и функция может быть как выпуклой вверх так и вниз. Это соответствует случаю $ChangesCount = 0$ и $ChangeSign = 0$. Если $ChangesCount = 0$ и $ChangeSign = -1$ ($ChangeSign = 1$), то функция $f(x)$ является выпуклой вверх (вниз), поэтому выдаем соответствующий ответ. В случае же когда $ChangesCount > 0$ ответ «не может быть выпуклой». Действительно, если это верно, то для всякой непрерывной $g(x)$ существуют такие ее точки $d_1.x < d_2.x \leq \ldots \leq d_5.x < d_6.x$ (две из них могут совпасть) что выполняется (без ограничения общности рассмотрим одни случай, остальные - аналогично): $g(x_2) < g(\frac{x_2-x_1}{x_3-x_1} \cdot x_1 + \frac{x_3-x_2}{x_3-x_1} \cdot x_3)$ и $g(x_5) > g(\frac{x_5-x_4}{x_6-x_4} \cdot x_4 + \frac{x_6-x_4}{x_6-x_5} \cdot x_6)$, что для выпуклой функции невозможно.

Время работы. Алгоритм сортирует массив точек по x за $O(n \log n)$, построение массива K и пробег по нему занимает $O(n)$ действий, поэтому итоговое время работы - $O(n \log n)$. □

Задача 6 (5). Дано полное тернарное дерево (корневое дерево, у каждой внутренней вершины три ребёнка), все листья помечены 0 или 1, а каждая внутренняя вершина имеет метку, которую имеет большинство её детей.

На вход подается массив меток листьев, нужно определить метку корня. Доказать, что в худшем случае, чтоб определить метку корня, требуется узнать метки всех листьев дерева

Критерии.

+0,5 объяснено в чём трудность — пример тернарного дерева высоты 1, в котором не зная одного листа нельзя определить корень.

+1 объяснения, что для каждого n можно рекурсивно построить пример (само это высказывание не очень осмыслено, но это некоторое продвижение в решении задачи).

2 доказательство по индукции существования примера трудного дерева, однако из-за отсутсвие честной формулировки утверждения доказывается не то, что нужно для решения задачи.

Решение. Начнём решение с объяснения того, что необходимо и достаточно доказать для решения задачи. Нужно доказать, что для каждого алгоритма и для каждого h существует такая разметка листьев полного тернарного дерева высоты h, что для определения метки корня алгоритму не достаточно запросить метки $n-1$ листа ($n = 3^h$). То есть, метка корня меняется при изменении неизвестной метки листа. Отметим, что алгоритм детерминированно выбирает листья — выбор следующего листа зависит только от известных до этого шага меток (при фиксированном h).

Также обратим внимание, что алгоритму нельзя указывать какой именно лист он не будет просматривать при фиксированной раскраске — разным алгоритмам могут соответствовать разные непросмотренные листья. Поэтому недостаточно построить одно дерево для всех алгоритмов — по дереву идёт квантор существования, в зависимости от алгоритма.

Итак, мы будем доказывать, заявленное утверждение индукцией по h используя метод противника (стратегия противника строит искомое дерево для каждого алгоритма). База. При $h = 1$ утверждение очевидно: в качестве метки первого листа, противник сообщает алгоритму 0, в качестве второго — 1, после чего меняет ответ в зависимости от того, какую метку алгоритм присвоил корню.

Шаг индукции. Пусть теперь для каждой высоты $1, \ldots, h-1$ существует стратегия противника. При запросе метки листа, метки братьев которого неизвестны, противник сообщает алгоритму 0; если известна метка только одного из братьев, то это метка 0, и противник сообщает алгоритму 1. Если же известны метки обоих братьев, то метка листа определят метку родителя. Противник сообщает её исходя из стратегии для дерева высоты $h-1$: родитель листа дерева высоты h — лист дерева высоты $h-1$.

□

Структуры данных

Задача 7 (3). Придя на безумное чаепитие, Алиса обнаружила, что пудинги там недостаточно сладкие. Алиса хочет, чтобы сладостность каждого пудинга была не меньше K. Для этих целей она смешивает пудинги по следующему правилу. Берёт два пудинга p_1 и p_2 с наименьшей сладостностью $s_1 \leqslant s_2$ и смешивает их. В результате получается пудинг со сладостностью $s = s_1 + 2s_2$, который занимает место пудинга p_1 (пудинг p_2 утилизирован). Постройте алгоритм, который получив на вход числа $n, K, s_1, \ldots s_n$, где n — число пудингов, возвращает последовательность пар номеров пудингов, которые необходимо смешать Алисе по её правилам для достижения сладостности не меньше K каждым из пудингов, или сообщает, что это сделать невозможно.

Указания. Нужно использовать кучу на минимум, последовательно извлекать два минимума, пока минимум больше K и добавлять новый элемент. Сложность $O(n \log n)$. □

Критерии.

0 вместо того, чтобы помогать Алисе исполнить её алгоритм, в решении явно строится собственный алгоритм.

0,5 отсортируем и будем идти слева направо (считалось, что смешав два пудинга с минимальной сладостностью получается пудинг с минимальной сладостностью).

1 решение через сортировку со вставками, однако считалось, что сложность $O(n \log n)$.

1,5 квадратичный алгоритм.

Алгоритмы на графах

Задача 8 (5). В графе G был проведён поиск в глубину. Время открытия и закрытия вершин сохранено в массивах d и f, а описание самого графа забыто. Приведите алгоритм, который находит корень дерева в лесе поиска в глубину, в котором лежит вершина v.

Критерии.

+1 Сформулировано неравенство для кандидата в корни (в духе $d[a] < d[b] < f[b] < f[a]$ с точностью до обозначений) /эквивалентная система неравенств/то же самое сказано словами.

+2 Описана процедура перехода к следующей вершине при подъеме к корню / описано нахождение ответа (для решений без итерации по кандидатам)

+1 Написано что-то разумное в духе обоснования корректности.

+1 Приведен линейный алгоритм и указана правильная асимптотика.

−2 В решении использована сортировка с более чем линейным временем работы.

Задача может быть решена в терминах вложенных отрезков. В таком случае балл выставляется на усмотрение проверяющего.

Решение. Для любого потомка b вершины a в лесе поиска в глубину справедливы следующие неравенства:

$$d[a] < d[b] < f[b] < f[a]. \tag{1}$$

Если вершина $u \neq v$ — корень дерева, в котором лежит v, то помимо неравенств (1) при $a = u$, $b = v$ должно быть выполнено следующее условие. Ни для какой вершины a не выполняются неравенства (1) при $b = u$. В противном случае, у корня окажется предок.

Алгоритм находит все вершины u, которые являются предками v, перебирая последовательно все вершины $u \in V$ и проверяя неравенства (1) при $a = u$, $b = v$. Если после нахождения одного предка u, был найден ещё один предок u', то u либо предок, либо потомок u'. Проверив это с

помощью неравенств (1) нужно оставить в качестве u вершину, которая является предком другой. Если же входе поиска не было найдено ни одной вершины u, то значит вершина v и была корнем своего дерева. Алгоритм работает за $O(|V|)$. □

Задача 9 (5). В стране произошло стихийное бедствие, в результате которого были разрушены многие дороги. Правительству нужно восстановить сообщение между городами (чтобы из каждого города можно было добраться до каждого); для этого нужно восстановить некоторые дороги (все они двусторонние) затратив на это минимум средств — восстановление i-ой дороги стоит c_i. Вход задачи: граф дорог страны; у каждого ребра-дороги указана цена восстановления; цена 0, если дорога не была разрушена. Постройте оптимальный алгоритм, который выводит список дорог, которые нужно восстановить для достижения цели, минимизируя затраты.

Критерии.

-5 рисует клику для конденсата;

-4 приведено только требование минимального остовного дерева MST без *правильного* обоснования;

$-0,5$ при условии выше - неправильно дана оценка на алгоритм построения MST;

-3 написано, что нужно именно дерево с минимальной стоимостью без обоснования почему именно дерево (и почему нельзя лучше);

-2 верный алгоритм с корректным обоснованием, но не оптимальный — работает за квадрат или куб;

-1 верный алгоритм с корректным обоснованием, но оценка на время работы посчитана неверно (или другой мелкий недочет).

Решение. Переформулируем задачу на языке теории графов. Нужно сделать граф связным, добавив рёбра минимальной суммарной стоимости. Достаточно сделать граф минимально-связным, то есть сделать из него дерево путём добавления рёбер, при этом сумма весов добавленных рёбер должна быть минимальной. Это тоже самое, что найти все рёбра

ненулевого веса некоторого минимального остовного дерева на исходном графе с указанными весами: если ребро стоит 0, то его добавление будет бесплатным, а значит, при использовании алгоритма Крускала, после добавления всех рёбер веса ноль, будут построены все компоненты связности, которые остались после землетрясения, а значит далее алгоритм Крускала найдёт все требуемые рёбра, посколько их суммарный вес будет минимальным из возожных по определению минимального остовного дерева.

Итак, мы доказали, что алгоритм, решающий задачу — алгоритм Крускала (или алгоритм Прима), запущенный на исходном графе дорог с весами после бедствия. Сложность алгоритма: $O(|E| \log |V|)$. □

Замечание. В этой задаче особенно ярко проявилось отсутствие обоснования корректности. Даже если вы поняли, что для решения задачи нужно построить минимальное остовное дерево, то этого ещё недостаточно: в качестве доказательства корректности необходимо было показать, что ненулевые рёбра минимального остовного дерева и есть искомые рёбра. Просто догадки недостаточно.

Задача 10 (2+5). Необходимо найти в ориентированном графе $G(V, E)$ с весами на рёбрах (быть может отрицательными, но циклов отрицательного веса в G нет) такие вершины u и v, что v достижима из u и стоимость $d[u, v]$ самого дешёвого пути из u в v максимальна. Постройте алгоритм, решающий задачу, при условии, что граф G

а) произвольный;

б) DAG и алгоритм работает за $O(|V|+|E|)$; в случае худшей асимптотики, задача оценивается в 2 очка.

Указания. а) Запускаем Флойда-Уоршолла и затем находим в матрице максимальный элемент, меньший бесконечности. Асимптотика $O(V^3)$.

б) Наша ошибка. Алгоритма с указанной асимптотикой мы не знаем. За указанную асимптотику ищется длиннейший простой путь в DAG. Но и для диаметра в DAG можно получить асимптотику $O(VE)$ с помощью топологической сортировки + динамического программирования. □

Критерии. а)

2 решение через алгоритм Флойда-Уоршелла;

1 решение через V запусков алгоритма Беллмана-Форда;

0 за использование алгоритма Дейкстры: задачу с отрицательными весами рёбер этот алгоритм не решает.

За ошибки в асимптотике снимать до 1 балла.

б)

2 решение за $O(VE)$;

1 тем, кто нашёл за $O(E)$ кратчайшие пути от одной вершины до остальных;

0 в остальных случаях.

Задача 11 (3). Разработчик ПО для навигатора столкнулся со следующей проблемой. Ему предоставили граф автодорог Москвы, вершины которого — перекрёстки (или развязки), а рёбра — дороги, но некоторые повороты на перекрёстках запрещены: для каждой вершины указаны **пары смежных с ней рёбер, которые описывают запрещённый поворот.** Постройте алгоритм, который находит кратчайшее расстояние между двумя вершинами этого графа. Граф взвешенный с положительными весами на рёбрах.

Решение. Обратите внимание, я выделил в условии жирным, запрещённый поворот описывается не ребром, а парой рёбер! Т.е. могу ли я поехать по дороге зависит от того, по какой дороге я на текущий перекрёсток выехал.

Построим новый граф, вершины которого пары (перекрёсток, ребро-дорога по которой на него въехали); ребро исходного графа соединяет две вершины нового, только если не образует запрещённый поворот. В таком графе кратчайшие расстояния ищутся обычным алгоритмом Дейкстры, а сам этот граф можно не строить отдельно, а просто модифицировать исходный, в процессе исполнения алгоритма.

Для оценки асимптотики посмотрим на кол-во вершин в новом графе $|V`| = \sum_{v \in V} deg(v) = 2|E|$. Число рёбер будет равно $|E`| = \sum_{v \in V} deg^2(v) =$

$O(E^2)$ (на самом деле граф перекрёстков разреженный и на практике как правило оказывается, что $|E`|$ порядка E)

Итоговая асимптотика Дейкстры получается $O(|E`| \log |V`|)$.

Существует алгоритм решение этой задачи, не опирающийся на разреженность графа и гарантирующий асимптотику $O(E \log V)$. $\square$

Критерии.

0.5 Из решения видно, что студент понимает, что поворот состоит из пары рёбер.

-0.5 Есть проблемы с асимптотикой – снимать от 0 до 0.5 балла.

$\leqslant 1.5$ В решении не указывается явное построение графа, а из листинга Дейкстры неясно что число вершин в модифицированном графе изменится, что в одну и ту же вершину мы будем заходить несколько раз для её релаксации.

$\leqslant 1$ Неверно модифицирован граф, вместо запрещённых поворотов (пар рёбер) использовали запрещённые рёбра – не более одного балла.

$\leqslant 0.5$ Если при этом даже не понимаете, что дорога может быть односторонней – не более 0.5 балла.

Динамическое программирование

Задача 12(2+7). Рассмотрим следующую постановку дискретной задачи про рюкзак. На вход подаются числа W и n и n троек (w_i, c_i, k_i), где w_i и c_i — вес и стоимость i-го предмета соответственно, а k_i количество таких предметов в магазине (все числа положительные и целые); у двух разных троек пары (w_i, c_i) не совпадают. Необходимо собрать рюкзак максимальной стоимости, веса не более W.

1. Постройте алгоритм, решающий эту задачу за $O(WK)$, где $K = \sum_i k_i$.

2. Постройте алгоритм, решающий эту задачу за $O(Wn)$.

Указания. 1. Задача сводится к рюкзаку без повторений. □

Решение.

Будем добавлять типы предметы по одному и хранить стандартный массив $d[i][0..W]$, в ячейке $d[i][j]$ будет лежать максимальная стоимость, которую можем получить набрав предметов типов $1, \ldots, i$ суммарного веса j. Инициализация динамики $d[0][0..W] = 0, \ldots, 0$.

Посмотрим на формулу для динамики после добавления i-го элемента: $d[i][j] = max_{0 \leqslant k \leqslant k_i}(d[i-1][j - k \cdot w_i] + k \cdot c_i)$ (здесь и далее в формулах будем считать, что обращения к элементам массива с отрицательными индексами выдают $-\infty$). Заметим, что ячека массива $d[i][j]$ зависит только от ячеек массива $d[i-1][j_1]$, у которых $j_1 \equiv j \pmod{w_i}$. Давайте придумаем способ пересчета индексов для каждого остатка $\pmod{w_i}$ за линейное время, тогда в сумме алгоритм также будет линейным. Перепишем формулу для конкретного остатка, сжав все индексы с этим остатком в один массив, чтобы перейти от весов в формуле к количествам добавляемых элементов: $d[i][j] = max_{0 \leqslant k \leqslant k_i}(d[i-1][j-k] + k \cdot c_i) = j \cdot c_i + max_{0 \leqslant k \leqslant k_i}(d[i-1][j-k] + (k-j) \cdot c_i)$. Прибавим ко всем элементам массива $d[i-1][j] += j \cdot c_i$. Тогда формула превратится в $d[i][j] = j \cdot c_i + max_{0 \leqslant k \leqslant k_i} d[i-1][j-k]$. Понятно, что вся сложность в вычислении второго слагаемого в этой формуле. Применив в этом месте дерево отрезков можно получить асимптотику $O(nW \log W)$. Чтобы добиться линейной асимптотики создадим стэк, у которого можно удалять элементы и из начала и из конца (что-то типа deque из C++) будет добавлять в него элементы $d[i-1][j]$ по одному, и при добавлении выкидывать с конца стэка все элементы меньшие добавленного. Тогда в стэке

все элементы всегда будут лежать по убыванию. Также при добавлении $d[i-1][j]$ будем смотреть, что лежит в начале стэка, если там лежит $d[i-1][j_{old}]$, где $j_{old} < j - k_i$, то его тоже выкинем. Тогда в стэке после добавления $d[i-1][j]$ будут лежать ровно те элементы по которым мы берем максимум в формуле для $d[i][j]$, причем максимальный всегда будет лежать в начале стэка и мы можем его достать за $O(1)$. В сумме каждый элемент будет добавлен и удален не более одного раза, значит алгоритм линейный.

□

Задача 13 (4+1). Будем обозначать большими буквами символы, а малыми — строки. Правило переписывания $A \to y$ означает, что в строке $X_1X_2\dots X_n$ любой символ $X_i = A$ можно заменить на строку y. Необходимо проверить, можно ли путём последовательных применений правил переписываний (в некотором порядке) получить из символа S строку z, в случае если каждое правило имеет вид $A \to BC$ или $A \to B$. Вход задачи: символ S, строка z, число n и последовательность из n правил переписываний указанного вида. Постройте оптимальный алгоритм, решающий задачу. Является ли он полиномиальным?

Указания. Это фактически алгоритм Кока-Янгера Кассами. Для каждого разбиения строки на две непустые части проверяем $z = xy$ проверяем, что есть правило $S' \to XY$ и из X можно получить x, а из Y — y, где $S' = S$ или S' достижимо из S переходами по одному символу. □